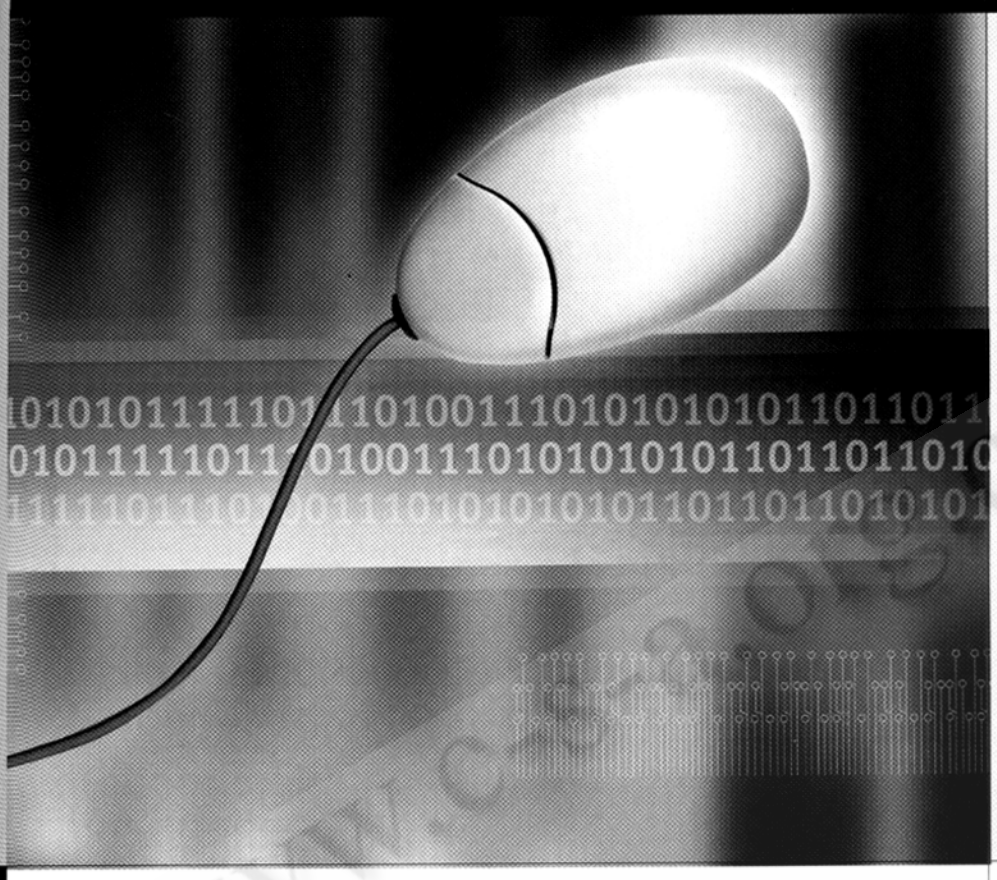
JAVA技术在远程实时数据的图像处理中的应用



The Application of the Java Technologies to The Image Processing of Remote Real-Time Data

左黎明　盛梅波　艾剑锋

(南昌华东交通大学基础科学学院　330003)

摘要：论述了如何利用Java Servlet相关技术实现三层网络服务结构，并把这种技术应用到远程实时数据的图像处理过程中,同时阐述了实现过程中的关键技术和方法。

关键词：Servlet　Java　三层网络服务结构　双缓冲技术　JDBC　Applet

1 引言

从SUN公司开发出JAVA语言以来,JAVA技术在短短的几年间得到了迅速发展,被应用到工程技术的诸多方面。JAVA是完全面向对象的网络语言，具备"一次编译，到处运行"的跨平台的良好特点，JAVA技术在网络上的应用前途十分光明。JAVA虽然是一门新兴的语言，但是它的某些技术，如Java Web Server、Java Servlet、Java RMI、JDBC等，已经很成熟了，而且在不断地向前发展。在远程实时数据图像处理中，这些技术给我们带来了极大的便利，我们现在可以很容易应用这些技术实现远程实时数据的图像处理。

2 JAVA远程实时数据的图像处理的过程

2.1 JAVA远程实时数据的图像处理模型

从图1中可以看出,我们采用的是三层网络服务结构模型(严格讲应该是四层,为保证安全性加了签名验证层)。随着Web技术的深入发展，传统的C/S结构的企业级应用系统已逐渐为B／S／D的三层结构所代替，这在以Java技术为核心的应用中得到了充分的体现。JAVA Servlet技术推动了以Java为核心技术的企业级三层Web应用的发展，它最适合于开发与Web服务器紧密相关的中间层。

2.2 JAVA Servlet技术的三层结构

JAVA Servlet技术的三层结构通常包含Web浏览器、中间层和远程数据库服务器三个层次。

(1) Web浏览器：Web浏览器是三层结构中的第一个层次，利用Web浏览器作为客户端，使客户面对一个统一的应用界面。

(2) Servlet中间层：中间层是指运行在服务器中的，联系Web浏览器与后台数据库服务器的软件或程序模块。

(3) 远程数据库服务器：远程数据库服务器是用户存放数据信息的地方，中间层可以通过ODBC(对CGI中间层)、JDBC-ODBC或是JDBC(对Servlet中间层)来访问远程数据库。

3 JAVA远程实时数据的图像处理中的关键技术

3.1 应用Servlet技术构造中间层

采用三层结构可以达到瘦客户端，结构清晰，便于管理的目的。这种结构把复杂的服务交给中间层专门处理，用户不需要了解中间层，如果对系统要进行改进，只需要修改中间层。目前可用于实施中间层的技术包含CGI、Java及

Servlet等。由于Servlet由Web服务器进行加载，利用Java语言进行开发，它在性能、可靠性以及可移植性等方面均比CGI有了长足的进步，因此Servlet是目前最适合实现中间层的技术。我们将通过远程股票信息系统来阐述如何应用Servlet技术构造中间层。该远程股票信息系统采用三层结构，开发工具包括：JDK1.3、JSDK和JAVA WEB SERVER2.0，数据库服务器采用Microsoft SQL SERVER2000。中间层Servlet为JudgeServlet，系统的流程包含五个步骤：

(1) 客户端提出读取数据库数据申请，数据库数据申请提交给中间层的JudgeServlet；

(2) 对客户端的申请进行数字签名验证，确认其合法性；

(3) 中间层JudgeServlet构建一个SQL语句，将SQL语句提交给JDBC；

(4) 远程数据库服务器执行SQL语句并将结果返回给中间层JudgeServlet；

(5) 中间层JudgeServlet将结果返回给客户端的Applet；

(6) 客户端的Applet对返回结果进行图像处理。

3.1.1 远程数据库结构

我们要读取的是远程数据库中的一个表，在该表中包含以下字段(如表1)

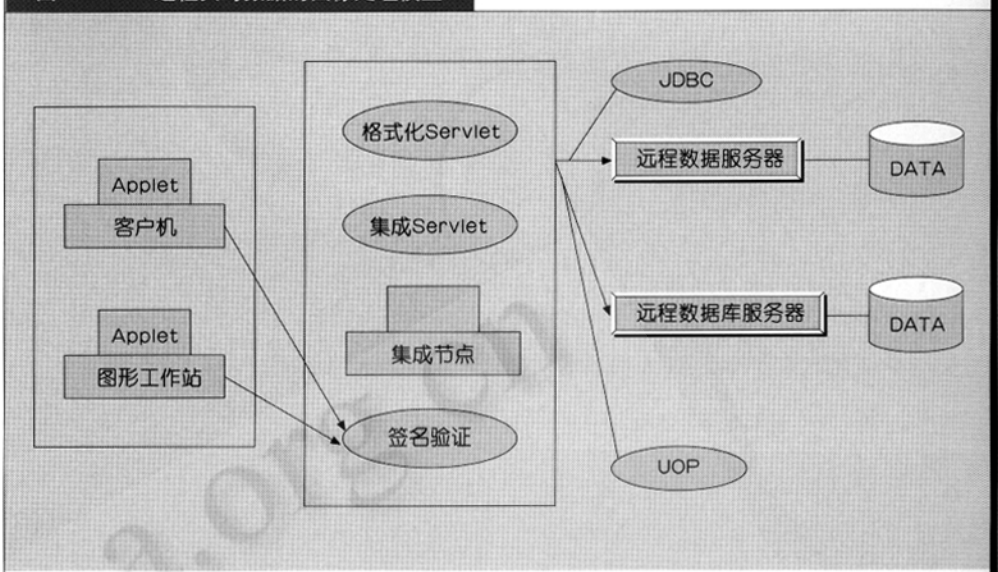
表1

列名称	类型	长度	说明	备注
Name	string	50	股票名称	NOT NULL
Price	integer	8	实时价格	NOT NULL

3.1.2 设计中间层JudgeServlet

用Servlet技术实现的中间层在客户端和远程数据库服务器之间建立一座安全的桥梁，我们将其命名为JudgeServlet，它根据客户端的Applet的数据请求来创建合适的SQL语句，将SQL语句提交给远程数据库服务器，然后根据SQL语句执行的结果传递给客户端的Applet。

图1 JAVA远程实时数据的图像处理模型



3.1.3 实现中间层

下面以JudgeServlet为例，说明如何实现中间层的Servlet。其中要用到JSDK中提供的GenericServlet类的子类HttpServlet。

初始化Servlet

```
public class JudgeServlet extends HttpServlet {
protected Connection dbConnection;
protected PrepareStatement readQuery;
protected PrepareStatement writeQuery;
protected String dbName=" jdbc:odbc:MONEY" ;  // 定义数据库名
protected String Name;  // 定义读取字段 Name
protected Integer Price;  // 定义读取字段 Price
public void init(ServletConfig config) throws ServletExecption
{/*init()方法获得一个servlet配置对象。这个对象在servlet引擎中执行，并允许servlet通过它获取相关参数 */
try {
Class.forName(" sun.jdbc.odbc.JdbcOdbcDriver" );
dbConnection=DriverManager.getConnection(dbName," " ," " ); // 连接数据库
}
catch(Exception e)
{
System.out.println(" Can not initialize database" );
}
```

Servlet的init()函数在Servlet被初次激活时进行调用，对于JudgeServlet，在init()中我们创建其与远程数据库的连接(在这里我们用的是JDBC-ODBC桥，当然远程数据库已经在ODBC中定义MONEY)，这里使用的是Java JDBC API中的Connection对象。

3.1.4 实现service()操作

当客户端Applet向Servlet进行请求时，Servlet的service()函数被调用，在service()中我们应该实现中间层的所有功能。

```
public void service(HttpServletRequest request，HttpServletResponse reponse)
throws ServletException,IOException
{// 使用 "request" 读取和请求有关信息，使用 "response" 返回数据
Name=request.getParameter(″ Name″ );
Price=request.getParameter(″ Price″ );
synchronized(this)// 对数据库查询，涉及资源共享，因此必须进行同步控制
{
doQuery (request,reponse);      // 执行要求的 SQL 操作
   ServletOutputStream out=reponse.getOutputStream();// 向客户端返回数据
.....
}}
```

首先，通过参数HttpServletRequest取得客户端的输入参数，这些参数由客户端的Applet产生的，然后根据需要进行库查询。查询数据库信息并向客户端Applet返回数据库查询结果。查询数据库信息首先应该根据查询条件，构建一个SQL语句，然后设置PrepaeStatement对象，通过运行其executeQuery()向后台数据库服务器请求结果。取得查询结果后，Servlet通过HTTPServletResponse生成一个结果返回给客户端Applet。

```
public void doQuery (HttpServletRequest request，HttpServletResponse reponse)
{ try {
readQuery=dbConnection.prepareStatement(″ SQL 指令″ );
ResultSet readResult=readQuery.executeQuery();
.....}

   catch(Exception e){ ... ... ... }
    }
```

3.2 客户端取得远程数据库中的实时数据

其主要方法是应用JAVA的线程机制，编制一个线程，每隔一定时间自动到相应结点读取最新的内容。假设远程服务器上存放的是某支股票的行情，并有程序按照一定的周期T1动态地更新其数据库中的内容，我们可以在Web中加入这种Java Applet，该Applet按照一定的周期T2向相应的中间层提出读取数据库数据的请求(需要注意的是必须保证T2<T1),这样一来可以让客户端得到动态的数据。而费时的数据处理交由Applet处理，将各时刻的数据绘制成图形，用动态变化的图形方式显示处理结果。

```
public class GetDyData extends java.applet.
Applet
    implements Runnable {
      Thread dthread;        // 定义一个线程.
        public void init()          // 初始化.
        {try
          { 连接中间件的代码段；}
        catch ( Malforme-dURLException e)
        {连接中间件失败处理的代码段；}
        }
        public void start() {
          if (dthread == null)
          {
          dthread = new Thread(this);
          dthread.start();
          }}
        public void stop()
        { if (dthread != null) {dthread.stop();
dthread = null;} }
        public void run() {
          InputStream filecon = null;
          DataInputStream filedata = null;
          while(true){
        try {向Servlet提出数据申请的代码段；
            接收返回数据流的代码段；
            }
        catch (IOException e)
        {System.out.println("Error in I/O:" +e.
getMessage());}
        try{dthread.sleep(500);}
        catch (InterruptedException e){}
        repaint();}
        } }
```

3.3 在客户端生成质量高的动态图形

在取得实时数据后，如何在客户端生成高质量的动态图形是一个需重点研究的问题。主要难点在于：

(1) 绘制每一帧图像花费的时间比较长(因为重绘图形时所要求的计算量很大)；

(2) 是在每次调用paint()前整个背景被清除，当在进行下一帧图像的计算时，用户看到的是背景，清除背景和绘制图形间的短暂时间被用户察觉，给人以闪烁感。我们通过重载update()方法并使用双缓冲技术很好的解决了闪烁问题。

3.3.1 重载 update()

当AWT接收到一个applet的重绘请求时，它就调用applet的update()。缺省地，update()清除applet的背景，然后调用paint()。重载update()，将以前在paint()中的绘图代码包含在update()中，从而避免每次重绘时将整个区域清除。既然背景不在自动清除，我们就需要自己在update()中完成。

3.3.2 双缓冲技术

主要原理是：创建一个后台图形对象，将一帧完整图像绘入该后台图形对象，然后调用drawImage()将整个图像一次性画到屏幕上去。采用双缓冲技术，大部分绘制是离屏的，将离屏图像一次性绘至屏幕上比直接在屏幕上绘制要有效得多，可以使动画平滑。使用双缓冲技术时，应重载update()方法。

如图2所示，在update()方法中，在后台缓冲区建立了一图形对象offScrGraphics:

```
Dimension d=size();
Image offScrImage=createImage(d.width,d.heigth);
Graphics offScrGraphics=offScrImage.getGraphics();
```

再将待显示的图像绘于offScrGraphics对象中：

```
offScrGraphics.drawImage(frames [I],0,0,this);
```

在paint()方法中，绘制好的offScrImage图像绘于前台缓冲区：

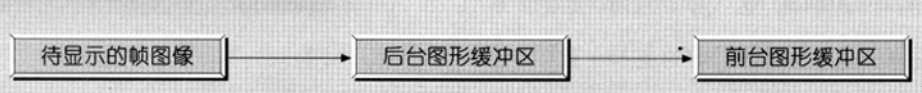
```
g.drawImage(offScrImage,0,0,this);
```

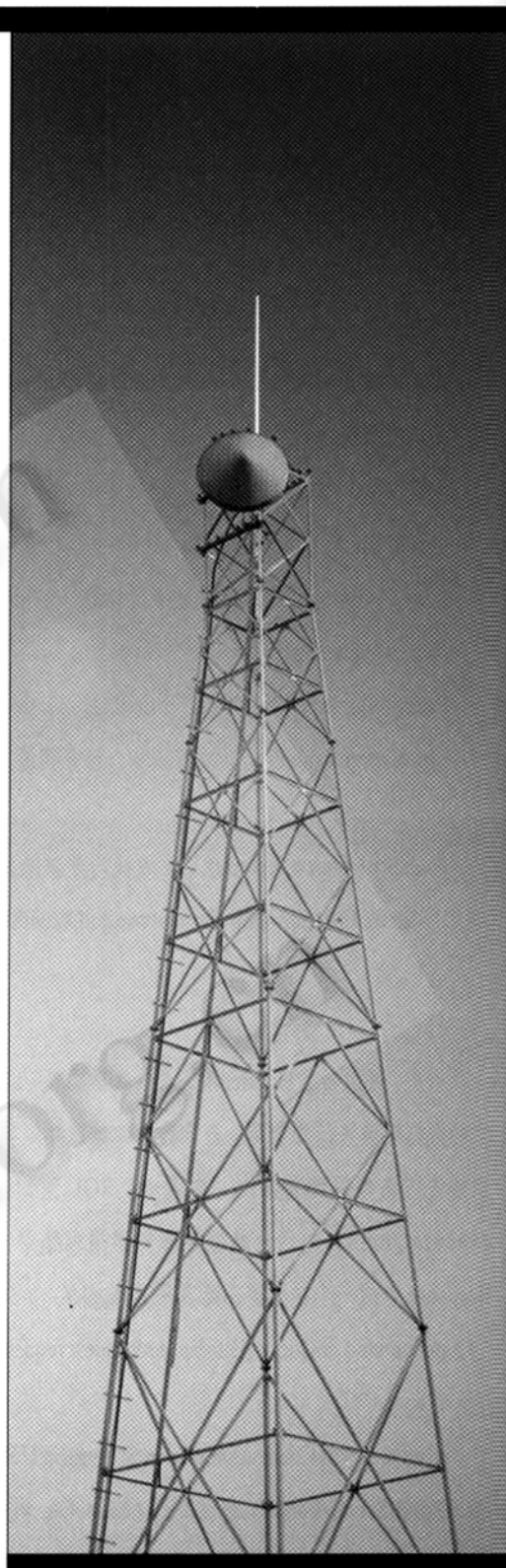
通过这种方法可以很好得解决图像闪烁的问题，而且不会影响到速度。

4 总结与探讨

在本文中通过Servlet利用Sockets与其他的Servlet或Applet进行通信的方法和JAVA的多线程机制来实现远程实时数据的图像处理。利用JAVA Servlet技术实现中间层是十分方便的，其操作过程类似于CGI编写的中间层。但由于Servlet是基于线程的，而且不会在每个用户进行请求时创建一个独立的进程完成操作(CGI是通过创建进程进行操作的)，因此其性能优于CGI。同时，由于Servlet是用Java语言进行编写的，因此其可移植性、代码重用性也超过CGI。对于如何对JAVA数据流进行签名和验证是一个很复杂的过程，今后将做进一步研究，这也是JAVA网络解决方案的一个重要的研究内容。

图2 双缓冲的过程





参考文献

1 宋辉、江峰等，JAVA服务器程序设计 [M]，清华大学出版社，2000。

2 刘丽玉、张龙祥等，JDBC与JAVA数据库程序设计 [M]，人民邮电出版社，2001。

3 李洪宝、曾文方等，基于web的实时信息发布系统的设计与实现，[J]，计算机应用，1999.12 27~28。